UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
10/730,900	12/10/2003	Jonathan Maron	19111.0117	5194

68009 7590 09/03/2008
Hanify & King, P.C.
1875 K Street
Suite 707
WASHINGTON, DC 20006

EXAMINER	
CHEN, QING	

ART UNIT	PAPER NUMBER
2191	

MAIL DATE	DELIVERY MODE
09/03/2008	PAPER

Please find below and/or attached an Office communication concerning this application or proceeding.

The time period for reply, if any, is set in the attached communication.

Office Action Summary	Application No.	Applicant(s)	
	10/730,900	MARON, JONATHAN	
	Examiner	Art Unit	
	Qing Chen	2191	

-- The MAILING DATE of this communication appears on the cover sheet with the correspondence address --

Period for Reply

A SHORTENED STATUTORY PERIOD FOR REPLY IS SET TO EXPIRE 3 MONTH(S) OR THIRTY (30) DAYS, WHICHEVER IS LONGER, FROM THE MAILING DATE OF THIS COMMUNICATION.

- Extensions of time may be available under the provisions of 37 CFR 1.136(a). In no event, however, may a reply be timely filed after SIX (6) MONTHS from the mailing date of this communication.
- If NO period for reply is specified above, the maximum statutory period will apply and will expire SIX (6) MONTHS from the mailing date of this communication.
- Failure to reply within the set or extended period for reply will, by statute, cause the application to become ABANDONED (35 U.S.C. § 133). Any reply received by the Office later than three months after the mailing date of this communication, even if timely filed, may reduce any earned patent term adjustment. See 37 CFR 1.704(b).

Status

1)☒ Responsive to communication(s) filed on *27 May 2008*.

2a)☒ This action is **FINAL**. 2b)☐ This action is non-final.

3)☐ Since this application is in condition for allowance except for formal matters, prosecution as to the merits is closed in accordance with the practice under *Ex parte Quayle*, 1935 C.D. 11, 453 O.G. 213.

Disposition of Claims

4)☒ Claim(s) *1,3-10,12-16,18-25,27-31,33-40 and 42-51* is/are pending in the application.

4a) Of the above claim(s) _____ is/are withdrawn from consideration.

5)☐ Claim(s) _____ is/are allowed.

6)☒ Claim(s) *1,3-10,12-16,18-25,27-31,33-40 and 42-51* is/are rejected.

7)☐ Claim(s) _____ is/are objected to.

8)☐ Claim(s) _____ are subject to restriction and/or election requirement.

Application Papers

9)☐ The specification is objected to by the Examiner.

10)☐ The drawing(s) filed on _____ is/are: a)☐ accepted or b)☐ objected to by the Examiner.

Applicant may not request that any objection to the drawing(s) be held in abeyance. See 37 CFR 1.85(a).

Replacement drawing sheet(s) including the correction is required if the drawing(s) is objected to. See 37 CFR 1.121(d).

11)☐ The oath or declaration is objected to by the Examiner. Note the attached Office Action or form PTO-152.

Priority under 35 U.S.C. § 119

12)☐ Acknowledgment is made of a claim for foreign priority under 35 U.S.C. § 119(a)-(d) or (f).

a)☐ All b)☐ Some * c)☐ None of:

1.☐ Certified copies of the priority documents have been received.

2.☐ Certified copies of the priority documents have been received in Application No. _____.

3.☐ Copies of the certified copies of the priority documents have been received in this National Stage application from the International Bureau (PCT Rule 17.2(a)).

* See the attached detailed Office action for a list of the certified copies not received.

Attachment(s)

1) ☐ Notice of References Cited (PTO-892)
2) ☐ Notice of Draftsperson's Patent Drawing Review (PTO-948)
3) ☐ Information Disclosure Statement(s) (PTO/SB/08) Paper No(s)/Mail Date _____.
4) ☐ Interview Summary (PTO-413) Paper No(s)/Mail Date. _____.
5) ☐ Notice of Informal Patent Application
6) ☐ Other: _____.

DETAILED ACTION

1. This Office action is in response to the amendment filed on May 27, 2008.

2. **Claims 1, 3-10, 12-16, 18-25, 27-31, 33-40, and 42-51** are pending.

3. **Claims 1, 4-6, 10, 12-16, 18-22, 24, 25, 27-31, 33-36, 40, and 42-51** have been amended.

4. **Claims 2, 11, 17, 26, 32, and 41** have been cancelled.

5. The objection to the drawings is withdrawn in view of Applicant's amendments to the specification.

6. The objection to the title is withdrawn in view of Applicant's amendments to the title.

7. The objections to Claims 1, 3-10, 12-15, 18, 27, 33, 42, 46, and 50 are withdrawn in view of Applicant's amendments to the claims.

8. The 35 U.S.C. § 112, second paragraph, rejections of Claims 1, 3-10, 12-16, 18-25, 27-31, 33-40, and 42-51 are withdrawn in view of Applicant's amendments to the claims.

Response to Amendment

Claim Rejections - 35 USC § 103

9. The following is a quotation of 35 U.S.C. 103(a) which forms the basis for all obviousness rejections set forth in this Office action:

> (a) A patent may not be obtained though the invention is not identically disclosed or described as set forth in section 102 of this title, if the differences between the subject matter sought to be patented and the prior art are such that the subject matter as a whole would have been obvious at the time the invention was made to a person having ordinary skill in the art to which said subject matter pertains. Patentability shall not be negatived by the manner in which the invention was made.

10. **Claims 1, 3-10, 12-16, 18-25, 27-31, 33-40, and 42-51** are rejected under 35 U.S.C. 103(a) as being unpatentable over **US 6,925,631 (hereinafter "Golden")** in view of **US 6,754,659 (hereinafter "Sarkar")** and **US 2003/0158832 (hereinafter "Sijacic")**.

As per **Claim 1**, Golden discloses:

- creating an event handler for a method node found in the markup language description *(see Column 9: 37-41, "The SAX parser, an event-driven API, is used for the parsing process. The application registers an event handler to a parser object that implements the org.sax.Parser interface. The event handler interface DocumentHandler is called whenever an element is found in the input stream.")*;

- registering the event handler *(see Column 9: 38-40, "The application registers an event handler to a parser object that implements the org.sax.Parser interface.")*;

- parsing the markup language description and invoking the registered event handler *(see Column 9: 53-61, "Upon parsing the input stream, a DOM representation of it is created. As an XML tag is found, an object (e.g. a JAVA class), as defined by the corresponding binding, is bound to the DOM tree for the specific tag. A tag with behavior (e.g. a JAVA class) bound to it is called a "taglet". As will be explained in more detail below, a init ( ) method is invoked on the taglet. After all of the taglet's children (possibly zero) have been added to the DOM representation, the taglet's run ( ) method is invoked.")*; and

- automatically generating output code using the invoked event handler *(see Column 14: 45-46, "... the taglet document is written to the output stream 15."; Column 17: 25-38, "FIG. 7 illustrates examples in which the above-described embodiments are used for*

transforming an XML input stream into a different output stream." and "The XBF engine 13 processes the XML input document 14 as described in the context of FIGS. 5 and 6, using bindings 12 which define the mapping between tags in the XML input document 14 and classes (e.g. JAVA classes), which give "behavior" to the tags. In one the examples depicted in FIG. 7, the "behavior" is the translation of the XML input document 14 into a HTML output document 50.").

However, Golden does not disclose:

- receiving an archive file to be deployed, wherein the archive file includes at least one input class;

- introspecting an input class included in the archive file to automatically generate information relating to the input class; and

- automatically generating a markup language description of the input class based on the generated information relating to the input class.

Sarkar discloses:

- receiving an archive file to be deployed, wherein the archive file includes at least one input class *(see Column 6: 50-55, "Then, at step 606, the Access Bean object AB701 retrieves environment information about generic EJB 720, locates it, and creates an instance of the generic EJB 720 using standard EJB API calls." It is inherent that the generic EJB includes at least one input class.)*; and

- introspecting an input class included in the archive file to automatically generate information relating to the input class *(see Column 6: 66 and 67 to Column 7: 1-4, "Specifically, at step 608, the generic EJB 720 uses Java class/introspection/reflection to retrieve the static*

variable of properties object C1 and obtain the name of the helper object it needs to instantiate which is based on the previously-set static variable." and 8-11, "At step 612, the generic EJB 720 uses Java reflection to get the main execution method of the helper object HO701 and, at step 614, invokes it, passing in the properties object C1 as an argument.").

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to incorporate the teaching of Sarkar into the teaching of Golden to include receiving an archive file to be deployed, wherein the archive file includes at least one input class; and introspecting an input class included in the archive file to automatically generate information relating to the input class. The modification would be obvious because one of ordinary skill in the art would be motivated to extract information from applications without modifying the applications themselves *(see Sarkar – Column 2: 10-13).*

Sijacic discloses:

- automatically generating a markup language description of the input class based on the generated information relating to the input class *(see Paragraph [0119], "Once a Java class that implements the ISimpleWorkPerformer interface is created and compiled, an XML description file for the class is defined. The XML description file specifies the environment, input, and output parameters that the class uses. In addition, the XML file specifies some optional design parameters that may control the custom activity's appearance in the process builder 391.").*

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to incorporate the teaching of Sijacic into the teaching of Golden to include automatically generating a markup language description of the input class based on the generated

information relating to the input class. The modification would be obvious because one of ordinary skill in the art would be motivated to describe a class of data objects using tags *(see Golden – Column 3: 40-48)*.

As per **Claim 3**, the rejection of **Claim 1** is incorporated; however, Golden does not disclose:

- extracting information identifying methods included in the input class; and
- for each method, extracting information relating to parameters of the method.

Sarkar discloses:

- extracting information identifying methods included in the input class *(see Column 6: 66 and 67 to Column 7: 1-4, "Specifically, at step 608, the generic EJB 720 uses Java class/introspection/reflection to retrieve the static variable of properties object C1 and obtain the name of the helper object it needs to instantiate which is based on the previously-set static variable." and 8-11, "At step 612, the generic EJB 720 uses Java reflection to get the main execution method of the helper object HO701 and, at step 614, invokes it, passing in the properties object C1 as an argument.")*; and
- for each method, extracting information relating to parameters of the method *(see Column 6: 66 and 67 to Column 7: 1-4, "Specifically, at step 608, the generic EJB 720 uses Java class/introspection/reflection to retrieve the static variable of properties object C1 and obtain the name of the helper object it needs to instantiate which is based on the previously-set static variable." and 8-11, "At step 612, the generic EJB 720 uses Java reflection to get the main*

execution method of the helper object HO701 and, at step 614, invokes it, passing in the properties object C1 as an argument.").

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to incorporate the teaching of Sarkar into the teaching of Golden to include extracting information identifying methods included in the input class; and for each method, extracting information relating to parameters of the method. The modification would be obvious because one of ordinary skill in the art would be motivated to extract information from applications without modifying the applications themselves *(see Sarkar – Column 2: 10-13).*

As per **Claim 4**, the rejection of **Claim 3** is incorporated; and Golden further discloses:

- automatically generating an Extensible Markup Language description of the input class based on the automatically generated information relating to the input class *(see Figure 1; Column 6: 36-38, "FIG. 1 shows an example of an extensible markup language input stream, here a fragment of a XML document which represents a small section of an order.").*

As per **Claim 5**, the rejection of **Claim 4** is incorporated; and Golden further discloses:

- creating a Simple Application Programming Interface for Extensible Markup Language event handler for a method node found in the Extensible Markup Language description *(see Column 9: 37-41, "The SAX parser, an event-driven API, is used for the parsing process. The application registers an event handler to a parser object that implements the org.sax.Parser interface. The event handler interface DocumentHandler is called whenever an element is found in the input stream.").*

As per **Claim 7**, the rejection of **Claim 1** is incorporated; and Golden further discloses:

- creating a plurality of event handlers for a method node found in the markup language description *(see Figure 2; Column 6: 51-61, "FIG. 2 illustrates that the tags of the XML fragment of FIG. 1 are mapped to classes of an object-oriented programming language by arrows pointing from each tag to a class. More specifically, the arrows point to "init" methods at the start-tags and to "run" methods at the end-tags.").*

As per **Claim 8**, the rejection of **Claim 7** is incorporated; and Golden further discloses:

- registering each of the plurality of event handlers *(see Column 9: 37-41, "The SAX parser, an event-driven API, is used for the parsing process. The application registers an event handler to a parser object that implements the org.sax.Parser interface. The event handler interface DocumentHandler is called whenever an element is found in the input stream.").*

As per **Claim 9**, the rejection of **Claim 8** is incorporated; and Golden further discloses:

- parsing the markup language description and invoking each of the plurality of registered event handlers *(see Column 9: 53-61, "Upon parsing the input stream, a DOM representation of it is created. As an XML tag is found, an object (e.g. a JAVA class), as defined by the corresponding binding, is bound to the DOM tree for the specific tag. A tag with behavior (e.g. a JAVA class) bound to it is called a "taglet". As will be explained in more detail below, a init ( ) method is invoked on the taglet. After all of the taglet's children (possibly zero) have been added to the DOM representation, the taglet's run ( ) method is invoked.").*

As per **Claim 10**, the rejection of **Claim 9** is incorporated; and Golden further discloses:

- automatically generating output code using each of the plurality of invoked event handler in parallel *(see Column 14: 45-46, "... the taglet document is written to the output stream 15."; Column 18: 12-13, "If the branching is not conditional, the two branches following the first engine work in parallel.").*

As per **Claim 12**, the rejection of **Claim 10** is incorporated; however, Golden does not disclose:

- extracting information identifying methods included in the input class; and
- for each method, extracting information relating to parameters of the method.

Sarkar discloses:

- extracting information identifying methods included in the input class *(see Column 6: 66 and 67 to Column 7: 1-4, "Specifically, at step 608, the generic EJB 720 uses Java class/introspection/reflection to retrieve the static variable of properties object C1 and obtain the name of the helper object it needs to instantiate which is based on the previously-set static variable." and 8-11, "At step 612, the generic EJB 720 uses Java reflection to get the main execution method of the helper object HO701 and, at step 614, invokes it, passing in the properties object C1 as an argument.")*; and
- for each method, extracting information relating to parameters of the method *(see Column 6: 66 and 67 to Column 7: 1-4, "Specifically, at step 608, the generic EJB 720 uses Java class/introspection/reflection to retrieve the static variable of properties object C1 and obtain*

the name of the helper object it needs to instantiate which is based on the previously-set static variable." and 8-11, "At step 612, the generic EJB 720 uses Java reflection to get the main execution method of the helper object HO701 and, at step 614, invokes it, passing in the properties object C1 as an argument.").

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to incorporate the teaching of Sarkar into the teaching of Golden to include extracting information identifying methods included in the input class; and for each method, extracting information relating to parameters of the method. The modification would be obvious because one of ordinary skill in the art would be motivated to extract information from applications without modifying the applications themselves *(see Sarkar – Column 2: 10-13).*

As per **Claim 13**, the rejection of **Claim 12** is incorporated; and Golden further discloses:

- automatically generating an Extensible Markup Language description of the input class based on the automatically generated information relating to the input class *(see Figure 1; Column 6: 36-38, "FIG. 1 shows an example of an extensible markup language input stream, here a fragment of a XML document which represents a small section of an order.").*

As per **Claim 14**, the rejection of **Claim 13** is incorporated; and Golden further discloses:

- creating a plurality of Simple Application Programming Interface for Extensible Markup Language event handlers for a method node found in the Extensible Markup Language description *(see Figure 2; Column 6: 51-61, "FIG. 2 illustrates that the tags of the XML fragment of FIG. 1 are mapped to classes of an object-oriented programming language by*

arrows pointing from each tag to a class. More specifically, the arrows point to "init" methods at the start-tags and to "run" methods at the end-tags.").

As per **Claim 16**, Golden discloses:

- a processor operable to execute computer program instructions *(see Column 7: 34-35, "... a computer system 10 with a processing unit and storage 11 for processing programs.")*;

- a memory operable to store computer program instructions executable by the processor *(see Column 7: 34-35, "... a computer system 10 with a processing unit and storage 11 for processing programs.")*; and

- computer program instructions stored in the memory and executable *(see Column 6: 25-28, "The disclosed embodiments of the computer program product comprise the disclosed program code which, for example, is stored on a computer-readable data carrier ...")* to perform the steps of:

- creating an event handler for a method node found in the markup language description *(see Column 9: 37-41, "The SAX parser, an event-driven API, is used for the parsing process. The application registers an event handler to a parser object that implements the org.sax.Parser interface. The event handler interface DocumentHandler is called whenever an element is found in the input stream.")*;

- registering the event handler *(see Column 9: 38-40, "The application registers an event handler to a parser object that implements the org.sax.Parser interface.")*;

- parsing the markup language description and invoking the registered event handler *(see Column 9: 53-61, "Upon parsing the input stream, a DOM representation of it is created.*

As an XML tag is found, an object (e.g. a JAVA class), as defined by the corresponding binding, is bound to the DOM tree for the specific tag. A tag with behavior (e.g. a JAVA class) bound to it is called a "taglet". As will be explained in more detail below, a init ( ) method is invoked on the taglet. After all of the taglet's children (possibly zero) have been added to the DOM representation, the taglet's run ( ) method is invoked."); and

- automatically generating output code using the invoked event handler *(see Column 14: 45-46, "... the taglet document is written to the output stream 15."; Column 17: 25-38, "FIG. 7 illustrates examples in which the above-described embodiments are used for transforming an XML input stream into a different output stream." and "The XBF engine 13 processes the XML input document 14 as described in the context of FIGS. 5 and 6, using bindings 12 which define the mapping between tags in the XML input document 14 and classes (e.g. JAVA classes), which give "behavior" to the tags. In one the examples depicted in FIG. 7, the "behavior" is the translation of the XML input document 14 into a HTML output document 50.").*

However, Golden does not disclose:

- receiving an archive file to be deployed, wherein the archive file includes at least one input class;

- introspecting an input class included in the archive file to automatically generate information relating to the input class; and

- automatically generating a markup language description of the input class based on the generated information relating to the input class.

Sarkar discloses:

- receiving an archive file to be deployed, wherein the archive file includes at least one input class *(see Column 6: 50-55, "Then, at step 606, the Access Bean object AB701 retrieves environment information about generic EJB 720, locates it, and creates an instance of the generic EJB 720 using standard EJB API calls." It is inherent that the generic EJB includes at least one input class.)*; and

- introspecting an input class included in the archive file to automatically generate information relating to the input class *(see Column 6: 66 and 67 to Column 7: 1-4, "Specifically, at step 608, the generic EJB 720 uses Java class/introspection/reflection to retrieve the static variable of properties object C1 and obtain the name of the helper object it needs to instantiate which is based on the previously-set static variable." and 8-11, "At step 612, the generic EJB 720 uses Java reflection to get the main execution method of the helper object HO701 and, at step 614, invokes it, passing in the properties object C1 as an argument.")*.

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to incorporate the teaching of Sarkar into the teaching of Golden to include receiving an archive file to be deployed, wherein the archive file includes at least one input class; and introspecting an input class included in the archive file to automatically generate information relating to the input class. The modification would be obvious because one of ordinary skill in the art would be motivated to extract information from applications without modifying the applications themselves *(see Sarkar – Column 2: 10-13)*.

Sijacic discloses:

- automatically generating a markup language description of the input class based on the generated information relating to the input class *(see Paragraph [0119], "Once a Java class*

that implements the ISimpleWorkPerformer interface is created and compiled, an XML description file for the class is defined. The XML description file specifies the environment, input, and output parameters that the class uses. In addition, the XML file specifies some optional design parameters that may control the custom activity's appearance in the process builder 391.").

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to incorporate the teaching of Sijacic into the teaching of Golden to include automatically generating a markup language description of the input class based on the generated information relating to the input class. The modification would be obvious because one of ordinary skill in the art would be motivated to describe a class of data objects using tags *(see Golden – Column 3: 40-48).*

As per **Claim 18**, the rejection of **Claim 16** is incorporated; however, Golden does not disclose:

- extracting information identifying methods included in the input class; and
- for each method, extracting information relating to parameters of the method.

Sarkar discloses:

- extracting information identifying methods included in the input class *(see Column 6: 66 and 67 to Column 7: 1-4, "Specifically, at step 608, the generic EJB 720 uses Java class/introspection/reflection to retrieve the static variable of properties object C1 and obtain the name of the helper object it needs to instantiate which is based on the previously-set static variable." and 8-11, "At step 612, the generic EJB 720 uses Java reflection to get the main*

execution method of the helper object HO701 and, at step 614, invokes it, passing in the properties object C1 as an argument."); and

- for each method, extracting information relating to parameters of the method *(see Column 6: 66 and 67 to Column 7: 1-4, "Specifically, at step 608, the generic EJB 720 uses Java class/introspection/reflection to retrieve the static variable of properties object C1 and obtain the name of the helper object it needs to instantiate which is based on the previously-set static variable." and 8-11, "At step 612, the generic EJB 720 uses Java reflection to get the main execution method of the helper object HO701 and, at step 614, invokes it, passing in the properties object C1 as an argument.")*.

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to incorporate the teaching of Sarkar into the teaching of Golden to include extracting information identifying methods included in the input class; and for each method, extracting information relating to parameters of the method. The modification would be obvious because one of ordinary skill in the art would be motivated to extract information from applications without modifying the applications themselves *(see Sarkar – Column 2: 10-13)*.

As per **Claim 19**, the rejection of **Claim 18** is incorporated; and Golden further discloses:

- automatically generating an Extensible Markup Language description of the input class based on the automatically generated information relating to the input class *(see Figure 1; Column 6: 36-38, "FIG. 1 shows an example of an extensible markup language input stream, here a fragment of a XML document which represents a small section of an order.")*.

As per **Claim 20**, the rejection of **Claim 19** is incorporated; and Golden further discloses:

- creating a Simple Application Programming Interface for Extensible Markup Language event handler for a method node found in the Extensible Markup Language description *(see Column 9: 37-41, "The SAX parser, an event-driven API, is used for the parsing process. The application registers an event handler to a parser object that implements the org.sax.Parser interface. The event handler interface DocumentHandler is called whenever an element is found in the input stream.").*

As per **Claim 31**, Golden discloses:

- a computer readable medium *(see Column 6: 25-28, "... a computer-readable data carrier ...")*; and

- computer program instructions, recorded on the computer readable medium, executable by a processor, *(see Column 6: 25-28, "The disclosed embodiments of the computer program product comprise the disclosed program code which, for example, is stored on a computer-readable data carrier ...")* for performing the steps of:

- creating an event handler for a method node found in the markup language description *(see Column 9: 37-41, "The SAX parser, an event-driven API, is used for the parsing process. The application registers an event handler to a parser object that implements the org.sax.Parser interface. The event handler interface DocumentHandler is called whenever an element is found in the input stream.")*;

- registering the event handler *(see Column 9: 38-40, "The application registers an event handler to a parser object that implements the org.sax.Parser interface.")*;

- parsing the markup language description and invoking the registered event handler *(see Column 9: 53-61, "Upon parsing the input stream, a DOM representation of it is created. As an XML tag is found, an object (e.g. a JAVA class), as defined by the corresponding binding, is bound to the DOM tree for the specific tag. A tag with behavior (e.g. a JAVA class) bound to it is called a "taglet". As will be explained in more detail below, a init ( ) method is invoked on the taglet. After all of the taglet's children (possibly zero) have been added to the DOM representation, the taglet's run ( ) method is invoked.")*; and

- automatically generating output code using the invoked event handler *(see Column 14: 45-46, "... the taglet document is written to the output stream 15."; Column 17: 25-38, "FIG. 7 illustrates examples in which the above-described embodiments are used for transforming an XML input stream into a different output stream." and "The XBF engine 13 processes the XML input document 14 as described in the context of FIGS. 5 and 6, using bindings 12 which define the mapping between tags in the XML input document 14 and classes (e.g. JAVA classes), which give "behavior" to the tags. In one the examples depicted in FIG. 7, the "behavior" is the translation of the XML input document 14 into a HTML output document 50.").*

However, Golden does not disclose:

- receiving an archive file to be deployed, wherein the archive file includes at least one input class;

- introspecting an input class included in the archive file to automatically generate information relating to the input class; and

- automatically generating a markup language description of the input class based on the generated information relating to the input class.

Sarkar discloses:

- receiving an archive file to be deployed, wherein the archive file includes at least one input class *(see Column 6: 50-55, "Then, at step 606, the Access Bean object AB701 retrieves environment information about generic EJB 720, locates it, and creates an instance of the generic EJB 720 using standard EJB API calls." It is inherent that the generic EJB includes at least one input class.)*; and

- introspecting an input class included in the archive file to automatically generate information relating to the input class *(see Column 6: 66 and 67 to Column 7: 1-4, "Specifically, at step 608, the generic EJB 720 uses Java class/introspection/reflection to retrieve the static variable of properties object C1 and obtain the name of the helper object it needs to instantiate which is based on the previously-set static variable." and 8-11, "At step 612, the generic EJB 720 uses Java reflection to get the main execution method of the helper object HO701 and, at step 614, invokes it, passing in the properties object C1 as an argument.")*.

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to incorporate the teaching of Sarkar into the teaching of Golden to include receiving an archive file to be deployed, wherein the archive file includes at least one input class; and introspecting an input class included in the archive file to automatically generate information relating to the input class. The modification would be obvious because one of ordinary skill in the art would be motivated to extract information from applications without modifying the applications themselves *(see Sarkar – Column 2: 10-13)*.

Sijacic discloses:

- automatically generating a markup language description of the input class based on the generated information relating to the input class *(see Paragraph [0119], "Once a Java class that implements the ISimpleWorkPerformer interface is created and compiled, an XML description file for the class is defined. The XML description file specifies the environment, input, and output parameters that the class uses. In addition, the XML file specifies some optional design parameters that may control the custom activity's appearance in the process builder 391.").*

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to incorporate the teaching of Sijacic into the teaching of Golden to include automatically generating a markup language description of the input class based on the generated information relating to the input class. The modification would be obvious because one of ordinary skill in the art would be motivated to describe a class of data objects using tags *(see Golden – Column 3: 40-48).*

As per **Claim 33**, the rejection of **Claim 31** is incorporated; however, Golden does not disclose:

- extracting information identifying methods included in the input class; and
- for each method, extracting information relating to parameters of the method.

Sarkar discloses:

- extracting information identifying methods included in the input class *(see Column 6: 66 and 67 to Column 7: 1-4, "Specifically, at step 608, the generic EJB 720 uses Java*

class/introspection/reflection to retrieve the static variable of properties object C1 and obtain the name of the helper object it needs to instantiate which is based on the previously-set static variable." and 8-11, "At step 612, the generic EJB 720 uses Java reflection to get the main execution method of the helper object HO701 and, at step 614, invokes it, passing in the properties object C1 as an argument."); and

- for each method, extracting information relating to parameters of the method *(see Column 6: 66 and 67 to Column 7: 1-4, "Specifically, at step 608, the generic EJB 720 uses Java class/introspection/reflection to retrieve the static variable of properties object C1 and obtain the name of the helper object it needs to instantiate which is based on the previously-set static variable." and 8-11, "At step 612, the generic EJB 720 uses Java reflection to get the main execution method of the helper object HO701 and, at step 614, invokes it, passing in the properties object C1 as an argument.").*

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to incorporate the teaching of Sarkar into the teaching of Golden to include extracting information identifying methods included in the input class; and for each method, extracting information relating to parameters of the method. The modification would be obvious because one of ordinary skill in the art would be motivated to extract information from applications without modifying the applications themselves *(see Sarkar – Column 2: 10-13).*

As per **Claim 34**, the rejection of **Claim 33** is incorporated; and Golden further discloses:

- automatically generating an Extensible Markup Language description of the input class based on the automatically generated information relating to the input class *(see Figure 1;*

Column 6: 36-38, "FIG. 1 shows an example of an extensible markup language input stream, here a fragment of a XML document which represents a small section of an order.").

As per **Claim 35**, the rejection of **Claim 34** is incorporated; and Golden further discloses:

- creating a Simple Application Programming Interface for Extensible Markup Language event handler for a method node found in the Extensible Markup Language description *(see Column 9: 37-41, "The SAX parser, an event-driven API, is used for the parsing process. The application registers an event handler to a parser object that implements the org.sax.Parser interface. The event handler interface DocumentHandler is called whenever an element is found in the input stream.").*

As per **Claim 37**, the rejection of **Claim 31** is incorporated; and Golden further discloses:

- creating a plurality of event handlers for a method node found in the markup language description *(see Figure 2; Column 6: 51-61, "FIG. 2 illustrates that the tags of the XML fragment of FIG. 1 are mapped to classes of an object-oriented programming language by arrows pointing from each tag to a class. More specifically, the arrows point to "init" methods at the start-tags and to "run" methods at the end-tags.").*

As per **Claim 38**, the rejection of **Claim 37** is incorporated; and Golden further discloses:

- registering each of the plurality of event handlers *(see Column 9: 37-41, "The SAX parser, an event-driven API, is used for the parsing process. The application registers an event*

handler to a parser object that implements the org.sax.Parser interface. The event handler interface DocumentHandler is called whenever an element is found in the input stream.").

As per **Claim 39**, the rejection of **Claim 38** is incorporated; and Golden further discloses:

- parsing the markup language description and invoking each of the plurality of registered event handlers *(see Column 9: 53-61, "Upon parsing the input stream, a DOM representation of it is created. As an XML tag is found, an object (e.g. a JAVA class), as defined by the corresponding binding, is bound to the DOM tree for the specific tag. A tag with behavior (e.g. a JAVA class) bound to it is called a "taglet". As will be explained in more detail below, a init ( ) method is invoked on the taglet. After all of the taglet's children (possibly zero) have been added to the DOM representation, the taglet's run ( ) method is invoked.").*

As per **Claim 40**, the rejection of **Claim 39** is incorporated; and Golden further discloses:

- automatically generating output code using each of the plurality of invoked event handler in parallel *(see Column 14: 45-46, "... the taglet document is written to the output stream 15."; Column 18: 12-13, "If the branching is not conditional, the two branches following the first engine work in parallel.").*

As per **Claim 42**, the rejection of **Claim 40** is incorporated; however, Golden does not disclose:

- extracting information identifying methods included in the input class; and
- for each method, extracting information relating to parameters of the method.

Sarkar discloses:

- extracting information identifying methods included in the input class *(see Column 6: 66 and 67 to Column 7: 1-4, "Specifically, at step 608, the generic EJB 720 uses Java class/introspection/reflection to retrieve the static variable of properties object C1 and obtain the name of the helper object it needs to instantiate which is based on the previously-set static variable." and 8-11, "At step 612, the generic EJB 720 uses Java reflection to get the main execution method of the helper object HO701 and, at step 614, invokes it, passing in the properties object C1 as an argument.")*; and

- for each method, extracting information relating to parameters of the method *(see Column 6: 66 and 67 to Column 7: 1-4, "Specifically, at step 608, the generic EJB 720 uses Java class/introspection/reflection to retrieve the static variable of properties object C1 and obtain the name of the helper object it needs to instantiate which is based on the previously-set static variable." and 8-11, "At step 612, the generic EJB 720 uses Java reflection to get the main execution method of the helper object HO701 and, at step 614, invokes it, passing in the properties object C1 as an argument.")*.

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to incorporate the teaching of Sarkar into the teaching of Golden to include extracting information identifying methods included in the input class; and for each method, extracting information relating to parameters of the method. The modification would be obvious because one of ordinary skill in the art would be motivated to extract information from applications without modifying the applications themselves *(see Sarkar – Column 2: 10-13)*.

As per **Claim 43**, the rejection of **Claim 42** is incorporated; and Golden further discloses:

- automatically generating an Extensible Markup Language description of the input class based on the automatically generated information relating to the input class *(see Figure 1; Column 6: 36-38, "FIG. 1 shows an example of an extensible markup language input stream, here a fragment of a XML document which represents a small section of an order.").*

As per **Claim 44**, the rejection of **Claim 43** is incorporated; and Golden further discloses:

- creating a plurality of Simple Application Programming Interface for Extensible Markup Language event handlers for a method node found in the Extensible Markup Language description *(see Figure 2; Column 6: 51-61, "FIG. 2 illustrates that the tags of the XML fragment of FIG. 1 are mapped to classes of an object-oriented programming language by arrows pointing from each tag to a class. More specifically, the arrows point to "init" methods at the start-tags and to "run" methods at the end-tags.").*

As per **Claim 46**, the rejection of **Claim 5** is incorporated; and Golden further discloses:

- registering the created Simple Application Programming Interface for Extensible Markup Language event handler for a method node found in the Extensible Markup Language description *(see Column 9: 37-41, "The SAX parser, an event-driven API, is used for the parsing process. The application registers an event handler to a parser object that implements the org.sax.Parser interface. The event handler interface DocumentHandler is called whenever an element is found in the input stream.").*

As per **Claim 6**, the rejection of **Claim 46** is incorporated; and Golden further discloses:

- parsing the Extensible Markup Language description using a Simple Application Programming Interface for Extensible Markup Language parser and invoking the Simple Application Programming Interface for Extensible Markup Language event handler *(see Column 9: 53-61, "Upon parsing the input stream, a DOM representation of it is created. As an XML tag is found, an object (e.g. a JAVA class), as defined by the corresponding binding, is bound to the DOM tree for the specific tag. A tag with behavior (e.g. a JAVA class) bound to it is called a "taglet". As will be explained in more detail below, a init ( ) method is invoked on the taglet. After all of the taglet's children (possibly zero) have been added to the DOM representation, the taglet's run ( ) method is invoked.").*

As per **Claim 47**, the rejection of **Claim 20** is incorporated; and Golden further discloses:

- registering the created Simple Application Programming Interface for Extensible Markup Language event handler for a method node found in the Extensible Markup Language description *(see Column 9: 37-41, "The SAX parser, an event-driven API, is used for the parsing process. The application registers an event handler to a parser object that implements the org.sax.Parser interface. The event handler interface DocumentHandler is called whenever an element is found in the input stream.").*

As per **Claim 21**, the rejection of **Claim 47** is incorporated; and Golden further discloses:

- parsing the Extensible Markup Language description using a Simple Application Programming Interface for Extensible Markup Language parser and invoking the Simple

Application Programming Interface for Extensible Markup Language event handler *(see Column 9: 53-61, "Upon parsing the input stream, a DOM representation of it is created. As an XML tag is found, an object (e.g. a JAVA class), as defined by the corresponding binding, is bound to the DOM tree for the specific tag. A tag with behavior (e.g. a JAVA class) bound to it is called a "taglet". As will be explained in more detail below, a init ( ) method is invoked on the taglet. After all of the taglet's children (possibly zero) have been added to the DOM representation, the taglet's run ( ) method is invoked.").*

As per **Claim 22**, the rejection of **Claim 21** is incorporated; and Golden further discloses:

- creating a plurality of event handlers for a method node found in the Extensible Markup Language description *(see Figure 2; Column 6: 51-61, "FIG. 2 illustrates that the tags of the XML fragment of FIG. 1 are mapped to classes of an object-oriented programming language by arrows pointing from each tag to a class. More specifically, the arrows point to "init" methods at the start-tags and to "run" methods at the end-tags.").*

As per **Claim 23**, the rejection of **Claim 22** is incorporated; and Golden further discloses:

- registering each of the plurality of event handlers *(see Column 9: 37-41, "The SAX parser, an event-driven API, is used for the parsing process. The application registers an event handler to a parser object that implements the org.sax.Parser interface. The event handler interface DocumentHandler is called whenever an element is found in the input stream.").*

As per **Claim 24**, the rejection of **Claim 23** is incorporated; and Golden further discloses:

- parsing the Extensible Markup Language description and invoking each of the plurality of registered event handlers *(see Column 9: 53-61, "Upon parsing the input stream, a DOM representation of it is created. As an XML tag is found, an object (e.g. a JAVA class), as defined by the corresponding binding, is bound to the DOM tree for the specific tag. A tag with behavior (e.g. a JAVA class) bound to it is called a "taglet". As will be explained in more detail below, a init ( ) method is invoked on the taglet. After all of the taglet's children (possibly zero) have been added to the DOM representation, the taglet's run ( ) method is invoked.").*

As per **Claim 25**, the rejection of **Claim 24** is incorporated; and Golden further discloses:

- automatically generating output code using each of the plurality of invoked event handler in parallel *(see Column 14: 45-46, "... the taglet document is written to the output stream 15."; Column 18: 12-13, "If the branching is not conditional, the two branches following the first engine work in parallel.").*

As per **Claim 27**, the rejection of **Claim 25** is incorporated; however, Golden does not disclose:

- extracting information identifying methods included in the input class; and
- for each method, extracting information relating to parameters of the method.

Sarkar discloses:

- extracting information identifying methods included in the input class *(see Column 6: 66 and 67 to Column 7: 1-4, "Specifically, at step 608, the generic EJB 720 uses Java class/introspection/reflection to retrieve the static variable of properties object C1 and obtain*

the name of the helper object it needs to instantiate which is based on the previously-set static variable." and 8-11, "At step 612, the generic EJB 720 uses Java reflection to get the main execution method of the helper object HO701 and, at step 614, invokes it, passing in the properties object C1 as an argument."); and

- for each method, extracting information relating to parameters of the method *(see Column 6: 66 and 67 to Column 7: 1-4, "Specifically, at step 608, the generic EJB 720 uses Java class/introspection/reflection to retrieve the static variable of properties object C1 and obtain the name of the helper object it needs to instantiate which is based on the previously-set static variable." and 8-11, "At step 612, the generic EJB 720 uses Java reflection to get the main execution method of the helper object HO701 and, at step 614, invokes it, passing in the properties object C1 as an argument.").*

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to incorporate the teaching of Sarkar into the teaching of Golden to include extracting information identifying methods included in the input class; and for each method, extracting information relating to parameters of the method. The modification would be obvious because one of ordinary skill in the art would be motivated to extract information from applications without modifying the applications themselves *(see Sarkar – Column 2: 10-13).*

As per **Claim 28**, the rejection of **Claim 27** is incorporated; and Golden further discloses:

- automatically generating an Extensible Markup Language description of the input class based on the automatically generated information relating to the input class *(see Figure 1;*

Column 6: 36-38, "FIG. 1 shows an example of an extensible markup language input stream, here a fragment of a XML document which represents a small section of an order.").

As per **Claim 29**, the rejection of **Claim 28** is incorporated; and Golden further discloses:

- creating a plurality of Simple Application Programming Interface for Extensible Markup Language event handlers for a method node found in the Extensible Markup Language description *(see Figure 2; Column 6: 51-61, "FIG. 2 illustrates that the tags of the XML fragment of FIG. 1 are mapped to classes of an object-oriented programming language by arrows pointing from each tag to a class. More specifically, the arrows point to "init" methods at the start-tags and to "run" methods at the end-tags.").*

As per **Claim 48**, the rejection of **Claim 35** is incorporated; and Golden further discloses:

- registering the created Simple Application Programming Interface for Extensible Markup Language event handler for a method node found in the Extensible Markup Language description *(see Column 9: 37-41, "The SAX parser, an event-driven API, is used for the parsing process. The application registers an event handler to a parser object that implements the org.sax.Parser interface. The event handler interface DocumentHandler is called whenever an element is found in the input stream.").*

As per **Claim 36**, the rejection of **Claim 48** is incorporated; and Golden further discloses:

- parsing the Extensible Markup Language description using a Simple Application Programming Interface for Extensible Markup Language parser and invoking the Simple

Application Programming Interface for Extensible Markup Language event handler *(see Column 9: 53-61, "Upon parsing the input stream, a DOM representation of it is created. As an XML tag is found, an object (e.g. a JAVA class), as defined by the corresponding binding, is bound to the DOM tree for the specific tag. A tag with behavior (e.g. a JAVA class) bound to it is called a "taglet". As will be explained in more detail below, a init ( ) method is invoked on the taglet. After all of the taglet's children (possibly zero) have been added to the DOM representation, the taglet's run ( ) method is invoked.").*

As per **Claim 49**, the rejection of **Claim 44** is incorporated; and Golden further discloses:

- registering the plurality of created Simple Application Programming Interface for Extensible Markup Language event handlers for a method node found in the Extensible Markup Language description *(see Column 9: 37-41, "The SAX parser, an event-driven API, is used for the parsing process. The application registers an event handler to a parser object that implements the org.sax.Parser interface. The event handler interface DocumentHandler is called whenever an element is found in the input stream.").*

As per **Claim 45**, the rejection of **Claim 49** is incorporated; and Golden further discloses:

- parsing the Extensible Markup Language description using a Simple Application Programming Interface for Extensible Markup Language parser and invoking the plurality of Simple Application Programming Interface for Extensible Markup Language event handlers *(see Column 9: 53-61, "Upon parsing the input stream, a DOM representation of it is created. As an XML tag is found, an object (e.g. a JAVA class), as defined by the corresponding binding, is*

bound to the DOM tree for the specific tag. A tag with behavior (e.g. a JAVA class) bound to it is called a "taglet". As will be explained in more detail below, a init ( ) method is invoked on the taglet. After all of the taglet's children (possibly zero) have been added to the DOM representation, the taglet's run ( ) method is invoked.").

As per **Claim 50**, the rejection of **Claim 14** is incorporated; and Golden further discloses:

- registering the plurality of created Simple Application Programming Interface for Extensible Markup Language event handlers for a method node found in the Extensible Markup Language description *(see Column 9: 37-41, "The SAX parser, an event-driven API, is used for the parsing process. The application registers an event handler to a parser object that implements the org.sax.Parser interface. The event handler interface DocumentHandler is called whenever an element is found in the input stream.").*

As per **Claim 15**, the rejection of **Claim 50** is incorporated; and Golden further discloses:

- parsing the Extensible Markup Language description using a Simple Application Programming Interface for Extensible Markup Language parser and invoking the plurality of Simple Application Programming Interface for Extensible Markup Language event handlers *(see Column 9: 53-61, "Upon parsing the input stream, a DOM representation of it is created. As an XML tag is found, an object (e.g. a JAVA class), as defined by the corresponding binding, is bound to the DOM tree for the specific tag. A tag with behavior (e.g. a JAVA class) bound to it is called a "taglet". As will be explained in more detail below, a init ( ) method is invoked on the*

taglet. After all of the taglet's children (possibly zero) have been added to the DOM representation, the taglet's run ( ) method is invoked.").

As per **Claim 51**, the rejection of **Claim 29** is incorporated; and Golden further discloses:

- registering the plurality of created Simple Application Programming Interface for Extensible Markup Language event handlers for a method node found in the Extensible Markup Language description *(see Column 9: 37-41, "The SAX parser, an event-driven API, is used for the parsing process. The application registers an event handler to a parser object that implements the org.sax.Parser interface. The event handler interface DocumentHandler is called whenever an element is found in the input stream.").*

As per **Claim 30**, the rejection of **Claim 51** is incorporated; and Golden further discloses:

- parsing the Extensible Markup Language description using a Simple Application Programming Interface for Extensible Markup Language parser and invoking the plurality of Simple Application Programming Interface for Extensible Markup Language event handlers *(see Column 9: 53-61, "Upon parsing the input stream, a DOM representation of it is created. As an XML tag is found, an object (e.g. a JAVA class), as defined by the corresponding binding, is bound to the DOM tree for the specific tag. A tag with behavior (e.g. a JAVA class) bound to it is called a "taglet". As will be explained in more detail below, a init ( ) method is invoked on the taglet. After all of the taglet's children (possibly zero) have been added to the DOM representation, the taglet's run ( ) method is invoked.").*

Response to Arguments

11. Applicant's arguments filed on May 27, 2008 have been fully considered, but they are not persuasive.

In the Remarks, Applicant argues:

a) For example, claim 1 requires introspecting an input class included in the archive file to automatically generate information relating to the input class. As the Examiner indicates, Golden does not disclose this requirement. Likewise, Sijacic and the combination of Golden and Sijacic do not disclose this requirement. Rather, the Examiner cites Sarkar at col. 6, line 66 to col. 7, line 4 as disclosing this requirement. However, as disclosed by Sarkar, JAVA class introspection is performed to retrieve the static variable of properties object C1 and obtain the name of the helper object it needs to instantiate. Thus, the introspection performed by Sarkar does not automatically generate information relating to an input class in an archive file, but merely allows the identification of an existing component to instantiate or invoke. This is different than the requirement of the claims to automatically generate information relating to the input class. As a result, the combination of Golden, Sarkar, and Sijacic does not disclose or suggest introspecting an input class included in the archive file to automatically generate information relating to the input class.

Examiner's response:

a) Examiner disagrees. Applicant's arguments are not persuasive for at least the following reasons:

First, with respect to the Applicant's assertion that Sarkar fails to disclose the limitation of "introspecting an input class included in the archive file to automatically generate information relating to the input class," as previously pointed out in the Non-Final Rejection (mailed on 11/28/2007) and further clarified herein, the Examiner respectfully submits that Sarkar clearly discloses "introspecting an input class included in the archive file to automatically generate information relating to the input class" *(see Column 6: 66 and 67 to Column 7: 1-4, "Specifically, at step 608, the generic EJB 720 uses Java class/introspection/reflection to retrieve the static variable of properties object C1 and obtain the name of the helper object it needs to instantiate which is based on the previously-set static variable." and 8-11, "At step 612, the generic EJB 720 uses Java reflection to get the main execution method of the helper object HO701 and, at step 614, invokes it, passing in the properties object C1 as an argument.")*. Note that the generic EJB uses Java™ introspection/reflection to retrieve the static variable of properties object C1 and the main execution method of the helper object HO701 (information relating to the input class).

Second, the Examiner further submits that one of ordinary skill in the computing art would readily recognize that the claimed archive file (*i.e.*, an Enterprise Java Bean (EJB™)) includes one or more classes of objects. As acknowledged by the Applicant in the "Background of the Invention" section of the specification, many Java™ technologies require the generation of classes based on some input class. An EJB™, for example, requires the generation of a particular application server's container classes to wrap the developer supplied bean class *(see Page 1, Lines 10-13)*.

Therefore, for at least the reasons set forth above, the rejections made under 35 U.S.C. § 103(a) with respect to Claims 1, 16, and 31 are proper and therefore, maintained.

In the Remarks, Applicant argues:

b) As a further example, claim 1 requires automatically generating a markup language description of the input class based on the generated information relating to the input class. The combination of Golden and Sarkar does not disclose or suggest this requirement. The Examiner cites Sijacic at para. 119 as disclosing this requirement. As disclosed by Sijacic, once a Java class that implements the ISimple-WorkPerformer interface is created and compiled, an XML description file for the class is defined that specifies the environment, input, and output parameters that the class uses and specifies some optional design parameters that may control the custom activity's appearance in the process builder 391. However, at para. [0098], Sijacic discloses that a Java editor and compiler may be used to create and compile a Java™ class that implements the ISimple-Workperformer interface. This makes it clear that activities described in paras. [0099] - [0149] are performed by a programmer using a text editor. That is, the code and the XML description file disclosed by Sijacic are typed in by a programmer during the design phase of the project, not automatically generated in order to deploy the software. As a result, the combination of Golden, Sarkar, and Sijacic does not disclose or suggest automatically generating a markup language description of the input class based on the generated information relating to the input class.

Examiner's response:

b) Examiner disagrees. Applicant's arguments are not persuasive for at least the following reasons:

First, with respect to the Applicant's assertion that Sijacic fails to disclose the limitation of "automatically generating a markup language description of the input class based on the generated information relating to the input class," as previously pointed out in the Non-Final Rejection (mailed on 11/28/2007) and further clarified herein, the Examiner respectfully submits that Sijacic clearly discloses "automatically generating a markup language description of the input class based on the generated information relating to the input class" *(see Paragraph [0119], "Once a Java class that implements the ISimpleWorkPerformer interface is created and compiled, an XML description file for the class is defined. The XML description file specifies the environment, input, and output parameters that the class uses. In addition, the XML file specifies some optional design parameters that may control the custom activity's appearance in the process builder 391.").* Note that an XML description file is defined for the Java™ class.

Second, with respect to the Applicant's assertion that activities described in paragraphs [0099]-[0149] of Sijacic are performed by a programmer using a text editor, the Examiner respectfully submits that there is no evidence, either explicitly or implicitly, in paragraphs [0118]-[0149] of Sijacic or in its entirety that suggests manually generating the XML description file. As pointed out by the Applicant, in paragraph [0098] of Sijacic, Sijacic discloses that "[t]o create the exemplary HelloWorldPerformer.java class, a Java™ editor and compiler may be used to create and compile a Java™ class that implements the ISimpleWorkPerformer interface." Examiner would like to point out that the disclosed Java™ editor and compiler is used to create the HelloWorldPerformer.java class, NOT the XML description file as averred by the Applicant.

Inasmuch as a Java™ editor and compiler may be used to create and compile a Java™ class that implements the ISimpleWorkPerformer interface, the Java™ editor and compiler are only associated with activities described in paragraphs [0099]-[0117] of Sijacic (*i.e.*, creating and compiling the HelloWorldPerformer.java class). In addition, one of ordinary skill in the computing art would readily recognize that a compiler involves an automated process in translating a computer program from a high-level programming language to a lower level language (*e.g.*, assembly language or machine language). Thus, the activities described in paragraphs [0099]-[0117] of Sijacic are not manually performed by a programmer using a text editor *per se* since a compiler is also involved in those activities.

Third, the Examiner further submits although Sijacic does not explicitly disclose an "automatic" generation of the XML description file, one of ordinary skill in the computing art would readily recognize that defining an XML description file based on a Java™ class is normally performed automatically via a software program. In a complex software platform such as the Internet Service Deployment Platform (ISDP) which involves integrated layers of software supporting application development, it would be highly inefficient for a programmer to hand-code an XML description file based on information from a Java™ file. Allowing a programmer to perform such a task would lead to loss of time, higher cost, and more errors. The XML description file shown in paragraph [0149] of Sijacic is merely a short and concise example to facilitate the understanding of the features and principles of Sijacic's invention. Its brevity does not imply that it is manually generated.

Fourth, Sijacic also discloses that the XML description file is defined after the Java™ class is created and compiled *(see Paragraph [0119])*. In other words, once the Java™ class is

compiled, it is translated into a machine language that is no longer in a human-readable format and thus, a programmer would not be able to hand-code the XML description file based on information from the compiled Java™ class.

Therefore, for at least the reasons set forth above, the rejections made under 35 U.S.C. § 103(a) with respect to Claims 1, 16, and 31 are proper and therefore, maintained.

In the Remarks, Applicant argues:

c) As a still further example, claim 1 requires automatically generating output code using the invoked event handler. Sarkar, Sijacic, and the combination of Sarkar and Sijacic do not disclose this requirement. Golden discloses processing an extensible markup language input stream using discrete software components mapped to tags contained in the input stream. In particular, at col. 14, lines 45-46, Golden discloses that a taglet document is written to the output stream 15. The Examiner cites this portion of Golden as disclosing automatically generating output code using the invoked event handler. However, the writing of the taglet document to the output stream is the last step in the functionality shown in Fig. 5. Fig. 5 and its corresponding description do not disclose or suggest code generation, but rather disclose execution of code during the processing of a tag. That is, this portion of Golden discloses invoking an existing registered code module to process a tag, then writing the processed tag to the output stream. This does not disclose or suggest automatically generating output code using the invoked event handler. As a result, the combination of Golden, Sarkar, and Sijacic does not disclose or suggest automatically generating output code using the invoked event handler.

Examiner's response:

c) Examiner disagrees. With respect to the Applicant's assertion that Golden fails to disclose the limitation of "automatically generating output code using the invoked event handler," as previously pointed out in the Non-Final Rejection (mailed on 11/28/2007) and further clarified herein, the Examiner respectfully submits that Golden clearly discloses "automatically generating output code using the invoked event handler" *(see Column 14: 45-46, "... the taglet document is written to the output stream 15."; Column 17: 25-38, "FIG. 7 illustrates examples in which the above-described embodiments are used for transforming an XML input stream into a different output stream." and "The XBF engine 13 processes the XML input document 14 as described in the context of FIGS. 5 and 6, using bindings 12 which define the mapping between tags in the XML input document 14 and classes (e.g. JAVA classes), which give "behavior" to the tags. In one the examples depicted in FIG. 7, the "behavior" is the translation of the XML input document 14 into a HTML output document 50.").* Note that a HTML output document is generated by translating the XML input document using bindings which define the mapping between the XML tags and Java™ classes (*i.e.*, taglets).

Therefore, for at least the reason set forth above, the rejections made under 35 U.S.C. § 103(a) with respect to Claims 1, 16, and 31 are proper and therefore, maintained.

Conclusion

12. **THIS ACTION IS MADE FINAL.** Applicant is reminded of the extension of time policy as set forth in 37 CFR 1.136(a).

A shortened statutory period for reply to this final action is set to expire THREE MONTHS from the mailing date of this action. In the event a first reply is filed within TWO MONTHS of the mailing date of this final action and the advisory action is not mailed until after the end of the THREE-MONTH shortened statutory period, then the shortened statutory period will expire on the date the advisory action is mailed, and any extension fee pursuant to 37 CFR 1.136(a) will be calculated from the mailing date of the advisory action. In no event, however, will the statutory period for reply expire later than SIX MONTHS from the mailing date of this final action.

13. Any inquiry concerning this communication or earlier communications from the Examiner should be directed to Qing Chen whose telephone number is 571-270-1071. The Examiner can normally be reached on Monday through Thursday from 7:30 AM to 4:00 PM. The Examiner can also be reached on alternate Fridays.

If attempts to reach the Examiner by telephone are unsuccessful, the Examiner's supervisor, Wei Zhen, can be reached on 571-272-3708. The fax phone number for the organization where this application or proceeding is assigned is 571-273-8300.

Any inquiry of a general nature or relating to the status of this application or proceeding should be directed to the TC 2100 Group receptionist whose telephone number is 571-272-2100.

Information regarding the status of an application may be obtained from the Patent Application Information Retrieval (PAIR) system. Status information for published applications may be obtained from either Private PAIR or Public PAIR. Status information for unpublished applications is available through Private PAIR only. For more information about the PAIR

system, see http://pair-direct.uspto.gov. Should you have questions on access to the Private PAIR system, contact the Electronic Business Center (EBC) at 866-217-9197 (toll-free).

/QC/
August 22, 2008

/Wei Y Zhen/

Supervisory Patent Examiner, Art Unit 2191